US 20050223221A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2005/0223221 A1**

Proudler et al. (43) **Pub. Date: Oct. 6, 2005**

(54) **APPARATUS AND METHOD FOR CREATING A TRUSTED ENVIRONMENT**

(76) Inventors: **Graeme John Proudler**, Bristol (GB); **Boris Balacheff**, Bristol (GB); **David Plaquin**, Bristol (GB)

Correspondence Address:
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400 (US)

(21) Appl. No.: **11/090,964**

(22) Filed: **Mar. 25, 2005**

Related U.S. Application Data

(63) Continuation-in-part of application No. 10/303,690, filed on Nov. 21, 2002.

(30) **Foreign Application Priority Data**

Nov. 22, 2001 (GB) 0127978.5

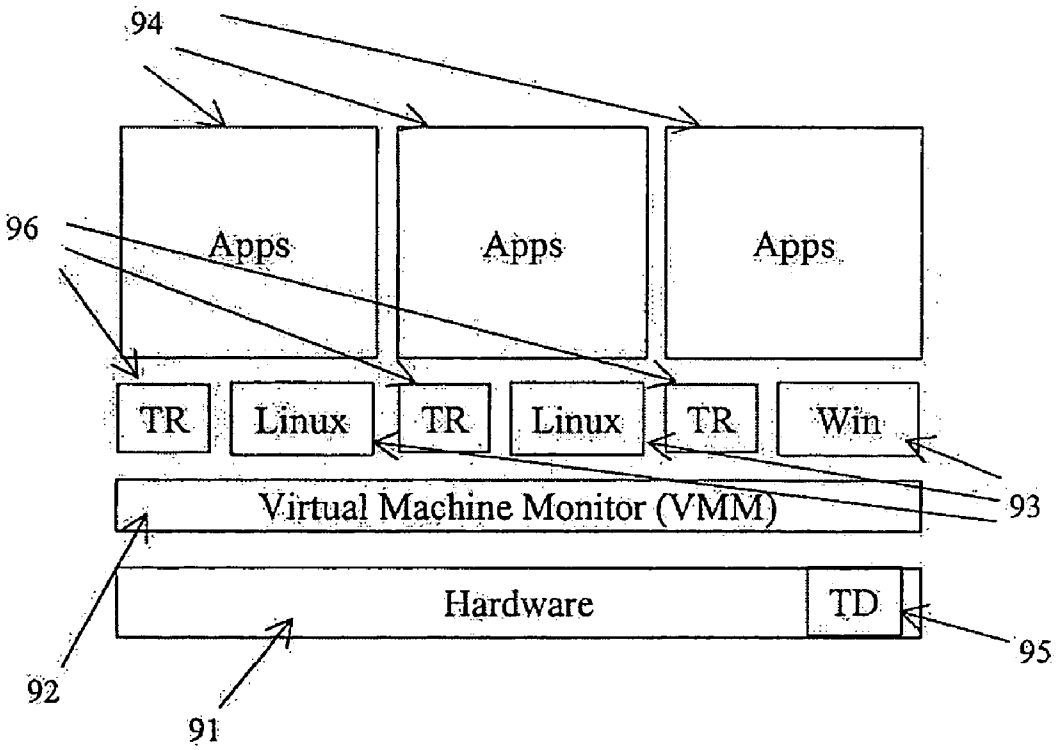
Publication Classification

(51) **Int. Cl.**[7] **G06F 9/45**

(52) **U.S. Cl.** **713/164**; 713/194

(57) **ABSTRACT**

A computer apparatus for creating a trusted environment comprising a trusted device arranged to acquire a first integrity metric to allow determination as to whether the computer apparatus is operating in a trusted manner; a processor arranged to allow execution of a first trust routine and associated first operating environment, and means for restricting the first operating environment access to resources available to the trust routine, wherein the trust routine being arranged to acquire the first integrity metric and a second integrity metric to allow determination as to whether the first operating environment is operating in a trusted manner.



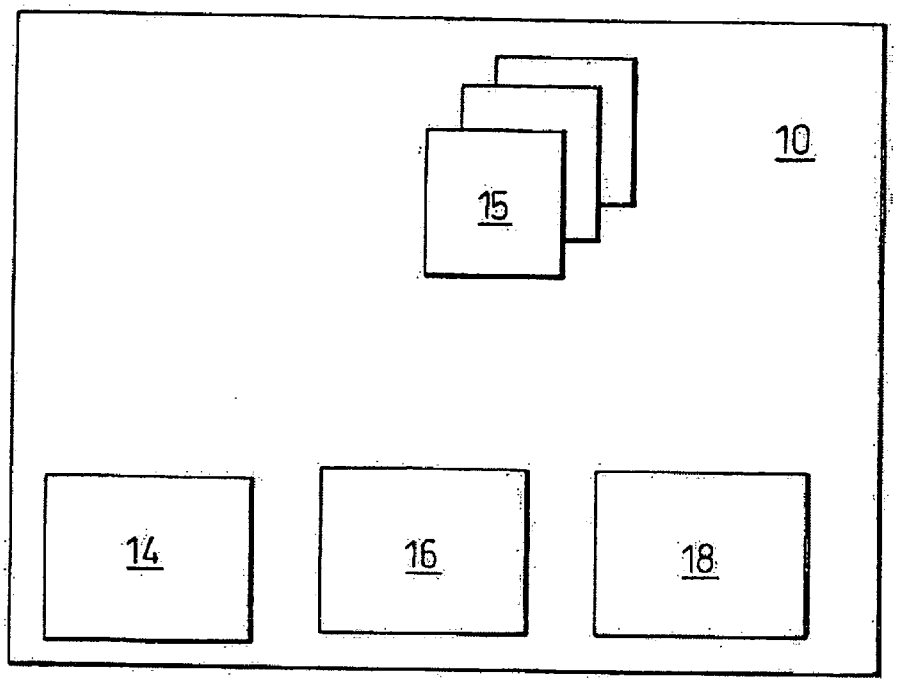


Fig. 1

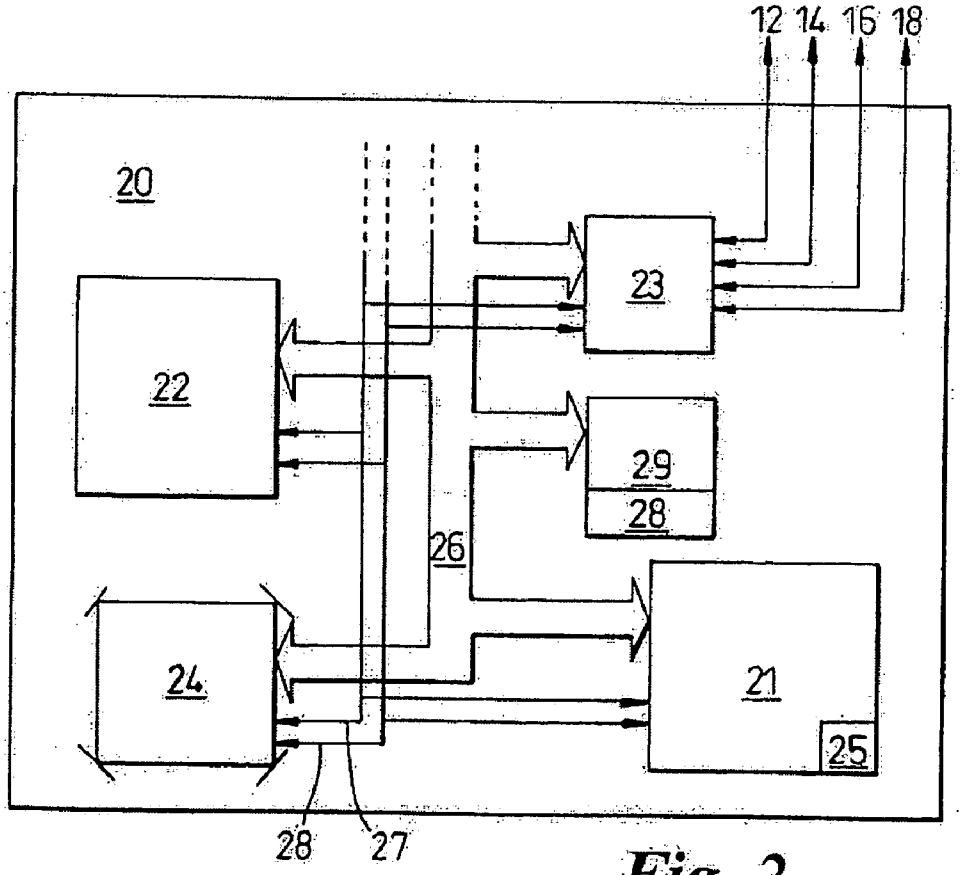


Fig. 2

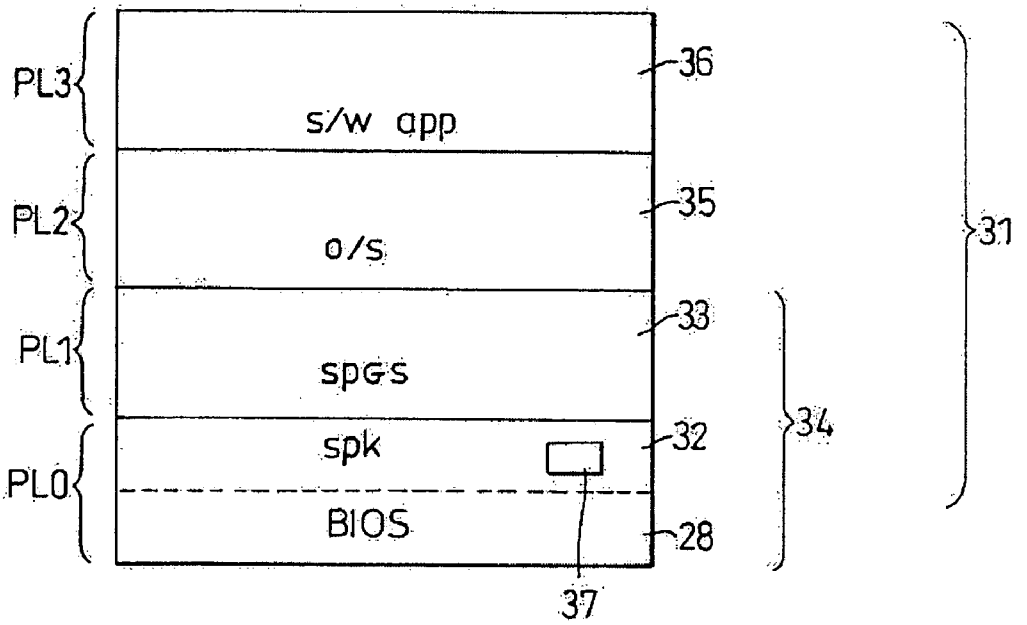


Fig. 3

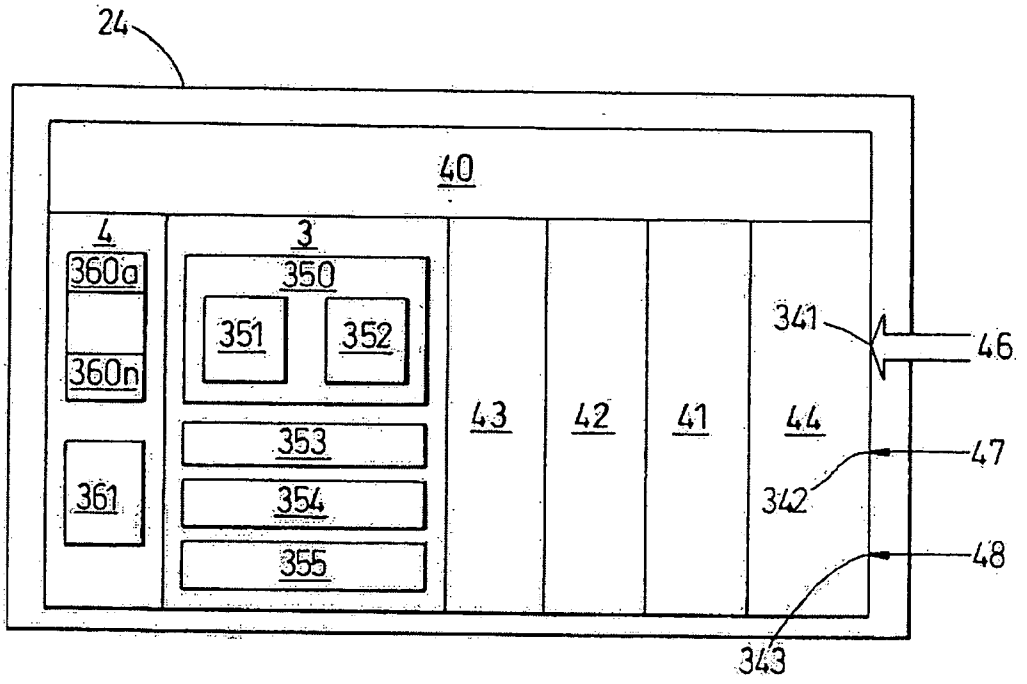

24
40
4
360a
360n
361
3
350
351
352
353
354
355
43
42
41
44
341
46
47
342
48
343

Fig. 4

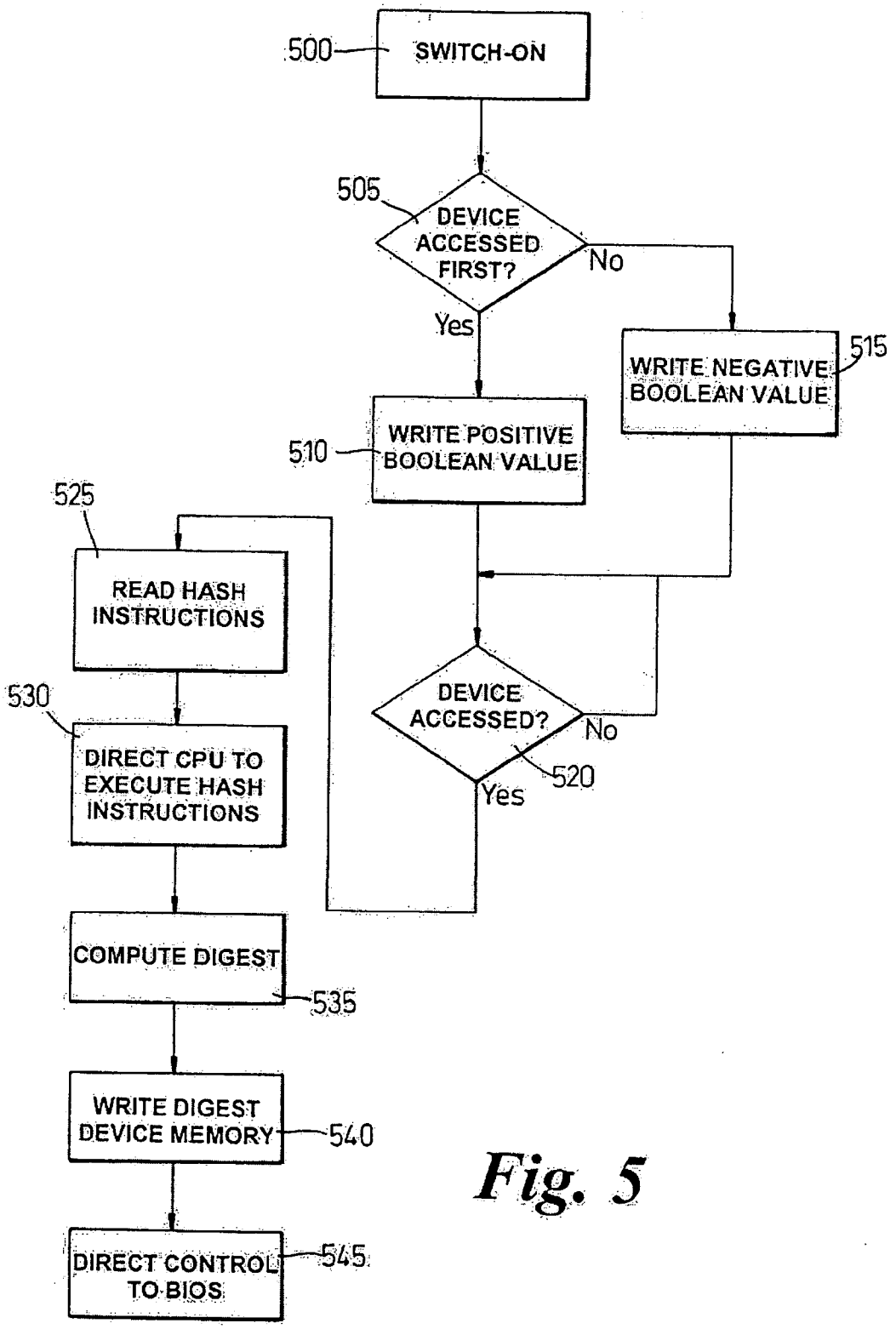

500
SWITCH-ON
505
DEVICE ACCESSED FIRST?
No
Yes
515
WRITE NEGATIVE BOOLEAN VALUE
510
WRITE POSITIVE BOOLEAN VALUE
525
READ HASH INSTRUCTIONS
530
DIRECT CPU TO EXECUTE HASH INSTRUCTIONS
DEVICE ACCESSED?
No
520
Yes
COMPUTE DIGEST
535
WRITE DIGEST DEVICE MEMORY
540
DIRECT CONTROL TO BIOS
545

Fig. 5

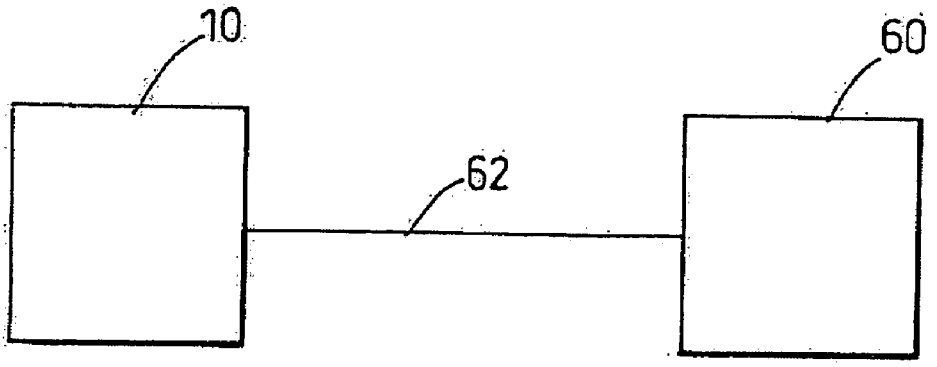


Fig. 6

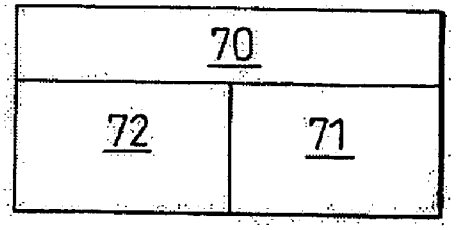


Fig. 7

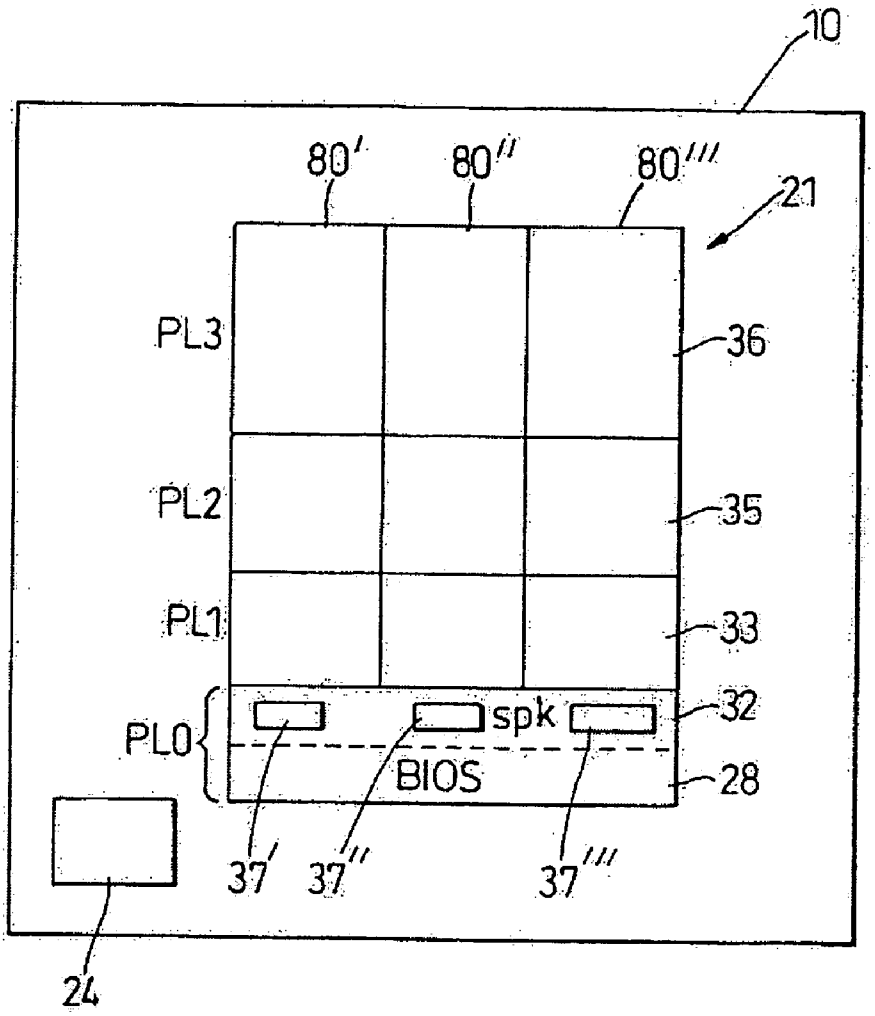


Fig. 8

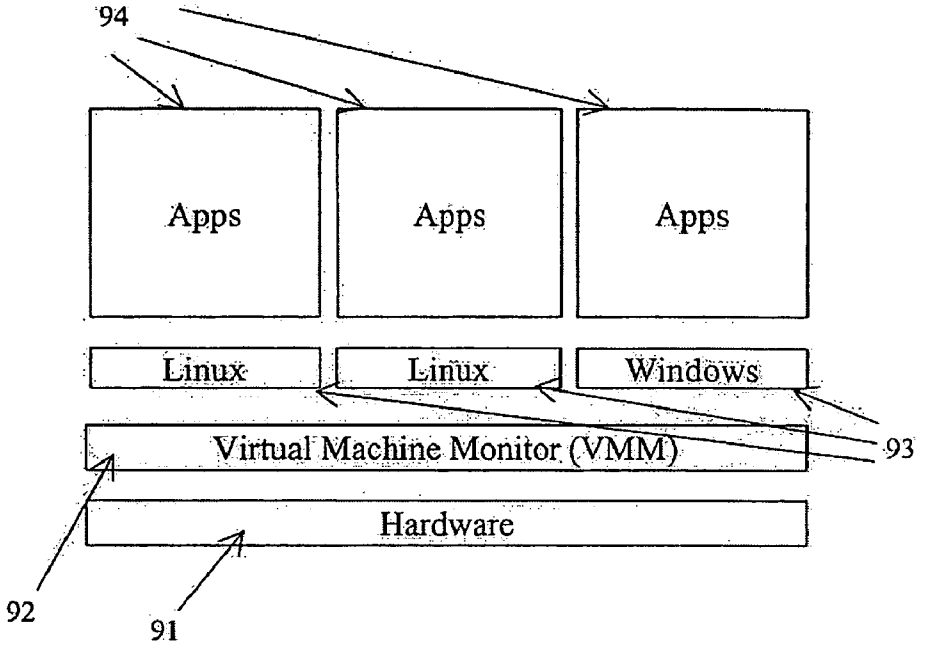


Figure 9

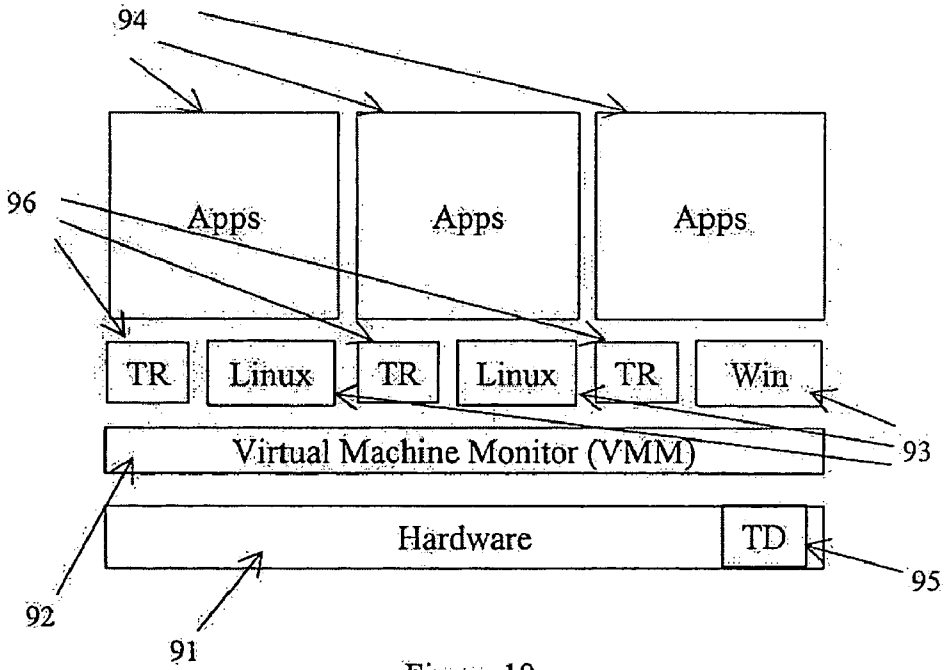


Figure 10

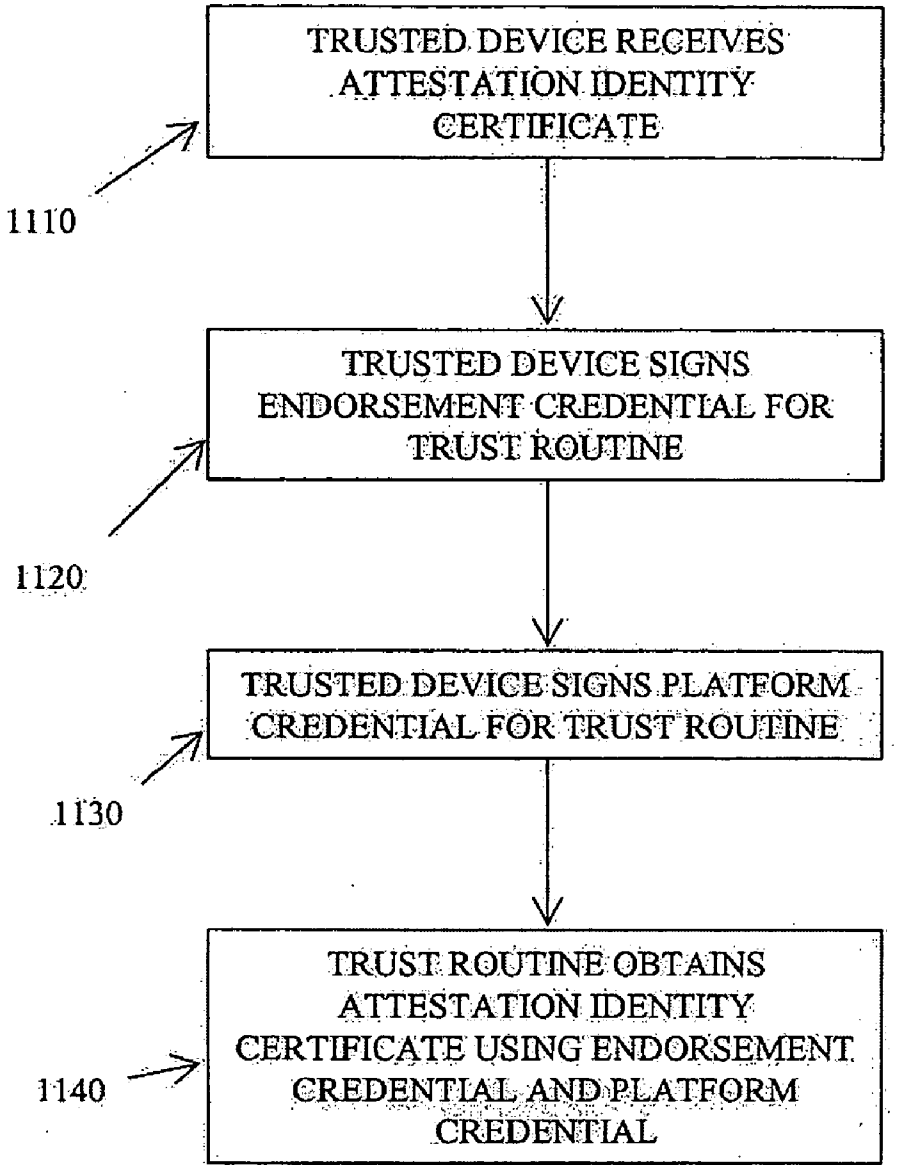

TRUSTED DEVICE RECEIVES ATTESTATION IDENTITY CERTIFICATE
1110
TRUSTED DEVICE SIGNS ENDORSEMENT CREDENTIAL FOR TRUST ROUTINE
1120
TRUSTED DEVICE SIGNS PLATFORM CREDENTIAL FOR TRUST ROUTINE
1130
TRUST ROUTINE OBTAINS ATTESTATION IDENTITY CERTIFICATE USING ENDORSEMENT CREDENTIAL AND PLATFORM CREDENTIAL
1140

Figure 11

APPARATUS AND METHOD FOR CREATING A TRUSTED ENVIRONMENT

FIELD OF THE INVENTION

[0001] The present invention relates to an apparatus and method for creating a trusted environment.

BACKGROUND

[0002] Computer platforms used for commercial applications typically operate in an environment where their behaviour is vulnerable to modification by local or remote entities.

[0003] Additionally, with the continuing increase in computer power it has become increasingly common for computer platforms to support multiple users, where each user can have their own operating environment installed on the computer platform. Various virtualization technologies have been developed to support this approach, typically allowing each user to have their own virtual machine running on the computer platform.

[0004] Where a number of separate operating systems are running simultaneously on a computer platform the operating systems are not necessarily isolated or protected from one another. The volume of source code for the software components involved is typically so large in modern operating systems that it is virtually impossible to ensure the correctness of the source code and whether the behaviour of the source code will behave as expected.

[0005] Accordingly, this potential insecurity of the platform is a limitation on its use by parties who might otherwise be willing to use the platform.

[0006] Increasing the level of trust in platforms therefore enables greater user confidence that the platform and operating system environment behave in a known manner.

SUMMARY OF THE INVENTION

[0007] In accordance with the present invention there is provided a computer apparatus for creating a trusted environment comprising a trusted device arranged to acquire a first integrity metric to allow determination as to whether the computer apparatus is operating in a trusted manner; a processor arranged to allow execution of a first trust routine and associated first operating environment, and means for restricting access of the first operating environment to resources available to the trust routine, wherein the trust routine is arranged to acquire the first integrity metric and a second integrity metric to allow determination as to whether the first operating environment is operating in a trusted manner.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] For a better understanding of the present invention and to understand how the same may be brought into effect reference will now be made, by way of example only, to the accompanying drawings, in which:—

[0009] **FIG. 1** illustrates a system capable of implementing embodiments of the present invention;

[0010] **FIG. 2** illustrates a motherboard for a computer platform of **FIG. 1** including a trusted device;

[0011] **FIG. 3** illustrates privilege levels of a processor useful for first embodiments of the present invention;

[0012] **FIG. 4** illustrates the trusted device of **FIG. 2** in more detail;

[0013] **FIG. 5** illustrates the steps involved in acquiring an integrity metric of the computing apparatus as used in embodiments of the invention;

[0014] **FIG. 6** illustrates a system capable of implementing first embodiments of the present invention;

[0015] **FIG. 7** illustrates a virtual trusted device in accordance with first embodiments of the present invention;

[0016] **FIG. 8** illustrates first embodiments of the present invention;

[0017] **FIG. 9** illustrates, generally, virtualization processes operating on computing apparatus;

[0018] **FIG. 10** illustrates the computer apparatus of **FIG. 10** modified according to second embodiments of the invention; and

[0019] **FIG. 11** illustrates a process of providing a trust routine with an attestation identity according to second embodiments of the invention.

SPECIFIC DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0020] Two sets of embodiments of the invention will now be described, with some common features (particularly in relation to basic platform architecture, trusted devices and collection of integrity metrics) between the two. The first set of embodiments describes a first approach to virtualization employing privilege levels of a processor to achieve isolation. The second set of embodiments describes another approach to virtualization and also indicates a form of virtual trusted device for a virtual operating environment developed to be highly consistent with Trusted Computing Group (TCG) specifications for trusted platforms and trusted devices.

[0021] The embodiments generally provide the incorporation into a computing platform of a physical trusted device and a software trust routine (i.e. a virtual trusted device). The function of the physical trusted device is to bind the identity of the platform to reliably measured data that provides an integrity metric of the platform, while the virtual trusted device binds the identity of an associated software operating environment (e.g. an operating system) to reliably measured data that provides an integrity metric of the operating environment. The identities and the integrity metrics may be compared with expected values provided by a trusted party (TP) that is prepared to vouch for the trustworthiness of the platform. Optionally, the expected values provided by the trusted third party are securely stored in the respective physical trusted device and the virtual trusted device. If there is a match, the implication is that at least part of the platform and operating system is operating correctly, depending on the scope of the integrity metric.

[0022] A user verifies the correct operation of the platform and operating environment before exchanging other data with the platform. A user does this by requesting the identities and integrity metrics of the physical trusted device and the virtual trusted device. (Optionally the trusted

devices will refuse to provide evidence of identity if it itself was unable to verify correct operation of the platform.) The user receives the proof of identity and the identity metric, and compares them against the values provided by the trusted third party. If the measured data reported by the trusted devices are the same as that provided by the trusted third party, the user can trust the platform.

[0023] Additionally, where the computer platform is arranged to support a plurality of separate operating environments, each operating environment having their own respective virtual trusted device, the users of the respective operating environments can trust that their operating environment is isolated from any other operating environment running on the computer platform.

[0024] Once a user has established trusted operation of the platform and operating environment, he exchanges other data with the platform. For a local user, the exchange might be by interacting with some software application running within the operating environment on the platform. For a remote user, the exchange might involve a secure transaction. In either case, the data exchanged is typically 'signed' by one of the trusted devices. The user can then have greater confidence that data is being exchanged with a platform whose behaviour can be trusted.

[0025] The trusted devices use cryptographic processes but do not necessarily provide an external interface to those cryptographic processes.

[0026] In first embodiments, to ensure there is a minimum risk that the virtual trusted device is susceptible to software attack by rogue software running on the computer platform the virtual trusted device is arranged to be executed in a processor privilege level that restricts access to other software applications being executed on the computer platform (as described below). Additionally, secrets associated with the virtual trusted device are stored such that the secrets are inaccessible to software applications being executed in a processor privilege level that is lower than that in which the virtual trusted device is executed. Also, a most desirable implementation would be to make the physical trusted device tamperproof, to protect secrets by making them inaccessible to other platform functions and provide an environment that is substantially immune to unauthorised modification. Since tamper-proofing is impossible, the best approximation is a trusted device that is tamper-resistant, or tamper-detecting. The trusted device, therefore, preferably consists of one physical component that is tamper-resistant.

[0027] Techniques relevant to tamper-resistance are well known to those skilled in the art of security. These techniques include methods for resisting tampering (such as appropriate encapsulation of the trusted device), methods for detecting tampering (such as detection of out of specification voltages, X-rays, or loss of physical integrity in the trusted device casing), and methods for eliminating data when tampering is detected.

[0028] A trusted platform **10** is illustrated in the diagram in **FIG. 1**. The platform **10** includes the standard features of a keyboard **14**, mouse **16** and visual display unit (VDU) **18**, which provide the physical 'user interface' of the platform. In the platform **10**, there are a plurality of modules **15**: these are other functional elements of the trusted platform of essentially any kind appropriate to that platform (the functional significance of such elements is not relevant to the present invention and will not be discussed further herein).

[0029] As illustrated in **FIG. 2**, the motherboard **20** of the trusted computing platform **10** includes (among other standard components) a main processor **21** with internal memory **25**, main memory **22**, a trusted device **24**, a data bus **26** and respective control lines **27** and lines **28**, BIOS memory **29** containing the BIOS program **28** for the platform **10** and an Input/Output (IO) device **23**, which controls interaction between the components of the motherboard, the keyboard **14**, the mouse **16** and the VDU **18**. The main memory **22** is typically random access memory (RAM).

[0030] In the first embodiment the processor **21** has four execution privilege levels PL**0**, PL**1**, PL**2**, PL**3**. Examples of such processors are the Hewlett-Packard's PA-RISC processor or Intel's IA-64 processor, however other processor configurations having a plurality of privilege levels can also be used.

[0031] Running in the processor **21** of this first embodiment is a secure platform architecture (SPA) **31**, as shown in **FIG. 3**.

[0032] The SPA **31** includes BIOS program or firmware **28** that runs on the processor **21** at execution privilege level **0** (PL**0**), the most privileged level of processor **21**. SPA **31** includes a four-layer software ring that runs on top of BIOS firmware **28** in processor **21**.

[0033] The innermost software ring, running on top of BIOS firmware **28**, is referred to as the secure platform kernel (SPK) **32** and is the only software ring that runs as a privileged task. SPK **32** runs at PL**0** and forms the foundation layer of SPA **31** and is the only ring layer that accesses privileged system registers and executes privileged instructions.

[0034] A secure platform global services module (SPGS) **33** runs on top of the SPK **32** as an unprivileged task. SPGS **33** runs at execution privilege level **1** (PL**1**), the second most privileged level of processor **21**. SPK **32** and SPKGS **33** are collectively referred to as secure platform (SP) **34**.

[0035] At least one operating system image **35** runs on top of SPGS **33** as an unprivileged task. Operating system image **35** runs at execution privilege level **2** (PL**2**), the third most privileged level of processor **31**. End user applications **36** run on top of operating system image(s) **35** as unprivileged tasks. End user applications **36** run at execution privilege level **3** (PL**3**), the fourth privileged level (i.e., the least privileged level) of processor **21**.

[0036] SPK **32** is preferably a small kernel of trusted, provably correct code that performs security critical services where the small size contributes to the SPK's security and correctness. Examples of security critical services include memory and process management, trap and interrupt handling, and cryptographic services, where some of these security services may be performed via a virtual trust device, as described below. SPGS **33** is constructed with trusted code, but utilizes hardware security capabilities of the processors **21**, such as IA-64 processors, to minimize the impact of a failure. SPGS **33** runs as an unprivileged task and employs SPK **32** to perform privileged operations.

[0037] Additionally, the SPK **32** includes code to allow execution of one or more virtual trusted devices **37** within

the SPK **32**. The virtual trusted device(s) **37** are associated with an operating environment executed in PL**2** and PL**3** and allow a user to establish whether the associated operating environment can be trusted, as described below. It is not essential, however, for the virtual trust device code to be incorporated within the SPK code, the code can be housed elsewhere, for example in the trusted device **24**.

[**0038**] To ensure that the virtual trusted device **37** can be trusted it is desirable for the manufacture of the SPK code to be validated by a trusted third party. On validation a validation credential signed with the trusted third parties private key is associated with the SPK code.

[**0039**] SPGS **33** typically includes all the services that do not have to be included in SPK **32**. One reason that secure platform **34** is split into SPK **32** and SPGS **33** is to permit SPK **32** to be small, stable and verifiable.

[**0040**] Interfaces between BIOS firmware **28** and processor hardware **21** include a privileged application binary interface (ABI) and a non-privileged ABI. The interfaces between SPK **32** and BIOS firmware **28** include a privileged ABI, a non-privileged ABI, and processor abstraction layer (PAL)/system abstraction layer (SAL)/extensible firmware interface (EFI) interfaces. The interfaces between SPGS **33** and SPK **32** include a secure platform interface (SPI) and a non-privileged ABI. The interfaces between operating system image(s) **35** and SPGS **33** include a SPI, a global services interface (GSI), and a non-privileged ABI. The interfaces between end user applications **36** and operating system image(s) **35** include an application program interface (API) and a non-privileged ABI.

[**0041**] SPGS **33** can partition operating system image layer **35** into multiple independent protection domains which operate at PL**2**. A protection domain is herein referred to as a software partition and associated collection of system resources, such as memory, I/O, processors, and the like, created by SPGS **33** for the purpose of loading and executing a single operating system image **35**. Each of the multiple independent protection domains are capable of booting and executing an operating system image **35** or any other program capable of operation using only SPK **32** and SPGS **33** services, such as a specialized application control program.

[**0042**] The multiple independent protection domains running at PL**2** are protected from each other through the memory protection capabilities of the four privilege level processor hardware **21**, such as the memory protection capabilities of the IA-64 processor. Therefore, a failure in one of the independent protection domains typically has no effect on the other independent protection domains, even if the failure is an operating system crash. The independent protection domains provide the capability to manage system utilization on a fine-grain basis while maintaining security. Operating system images **35** are ported to secure platform **34** of SPA **31** similar to how operating systems are ported to a new hardware platform industrial standard archicture ISA in the classical architecture for operating systems.

[**0043**] End user applications **36** run at the least privileged level, PL**3**, as unprivileged tasks under the control of an operating system image **35** in a secure platform **34** protection domain. Typically, from the end user application perspective, the end user application **36** operates under the control of an operating system image **35** as the end user application would run under the control of an operating system in the classical architecture for operating systems.

[**0044**] In order for the computer platform **10** and operating environment(s) to be trusted, a chain of trust from the system hardware, through the boot process, to final running code is established. In addition, all software code is preferably authenticated before being executed, and a properly authenticated piece of code is preferably unchangeable except by a similarly trusted component to maintain the chain of trust. The software authentication should be more than a simple check sum or other forgeable scheme. Thus, SPA **31** preferably employs strong authentication using cryptographic methods, such as public key encryption, such that software can be undetectably corrupt only if a private key is known.

[**0045**] The chain of trust extends back to the trusted device **24**. As described below, after system reset the processor **21** is initially controlled by the trusted device **24**, which then after performing a secure boot process hands control over to the BIOS firmware **28**. During the secure boot process, the trusted device **24** acquires an integrity metric of the computer platform **10**, as described below.

[**0046**] Specifically, the trusted device **24** used for embodiments of the invention comprises, as shown in **FIG. 4**: a controller **40** programmed to control the overall operation of the trusted device **24**, and interact with the other functions on the trusted device **24** and with the other devices on the motherboard **20**; a measurement function **41** for acquiring the integrity metric from the platform **10**; a cryptographic function **42** for signing, encrypting or decrypting specified data; an authentication function **43**; and interface circuitry **44** having appropriate ports (**46**, **47** & **48**) for connecting the trusted device **24** respectively to the data bus **26**, control lines **27** and address lines **28** of the motherboard **20**. Each of the blocks in the trusted device **24** has access (typically via the controller **40**) to appropriate volatile memory areas **4** and/or non-volatile memory areas **3** of the trusted device **24**. Additionally, the trusted device **24** is designed, in a known manner, to be tamper resistant.

[**0047**] For reasons of performance, the trusted device **24** may be implemented as an application specific integrated circuit (ASIC). However, for flexibility, the trusted device **24** is preferably an appropriately programmed micro-controller. Both ASICs and micro-controllers are well known in the art of microelectronics and will not be considered herein in any further detail.

[**0048**] One item of data stored in the non-volatile memory **3** of the trusted device **24** is a certificate **350**. The certificate **350** contains at least a public key **351** of the trusted device **24** and optionally an authenticated value **352** of the platform integrity metric measured by a trusted party (TP). The certificate **350** is signed by the TP using the TP's private key prior to it being stored in the trusted device **24**. In later communications sessions, a user of the platform **10** can verify the integrity of the platform **10** and operating environment by comparing the acquired integrity metric (i.e. measured integrity metric) with an authentic integrity metric **352**, as described below. Knowledge of the TP's generally-available public key enables simple verification of the certificate **350**. The non-volatile memory **45** also contains an identity (ID) label **353**. The ID label **353** is a conventional ID label, for example a serial number, that is unique within

some context. The ID label **353** is generally used for indexing and labelling of data relevant to the trusted device **24**, but is insufficient in itself to prove the identity of the platform **10** under trusted conditions.

[**0049**] The trusted third party that is requested to supply the authentic integrity metric will inspect the type of the platform to decide whether to vouch for it or not. This will be a matter of policy. If all is well the TP measures the value of integrity metric of the platform. Then, the TP generates a certificate for the platform. The certificate is generated by the TP by appending the trusted device's public key, and optionally its ID label, to the measured integrity metric, and signing the string with the TP's private key.

[**0050**] The trusted device **24** can subsequently prove its identity by using its private key to process some input data received from the user and produce output data, such that the input/output pair is statistically impossible to produce without knowledge of the private key. Hence, knowledge of the private key forms the basis of identity in this case. Clearly, it would be feasible to use symmetric encryption to form the basis of identity. However, the disadvantage of using symmetric encryption is that the user would need to share his secret with the trusted device. Further, as a result of the need to share the secret with the user, while symmetric encryption would in principle be sufficient to prove identity to the user, it would insufficient to prove identity to a third party, who could not be entirely sure the verification originated from the trusted device or the user.

[**0051**] The trusted device **24** is initialised by writing the certificate **350** into the appropriate non-volatile memory locations **3** of the trusted device **24**. This is done, preferably, by secure communication with the trusted device **24** after it is installed in the motherboard **20**. The method of writing the certificate to the trusted device **24** is analogous to the method used to initialise smart cards by writing private keys thereto. The secure communication is supported by a 'master key', known only to the TP, that is written to the trusted device (or smart card) during manufacture, and used to enable the writing of data to the trusted device **24**; writing of data to the trusted device **24** without knowledge of the master key is not possible.

[**0052**] At some later point during operation of the platform, for example when it is switched on or reset the trusted device **24** measures and stores the integrity metric **361** of the platform.

[**0053**] The trusted device **24** is equipped with at least one method of reliably measuring or acquiring the integrity metric of the computing platform **10** with which it is associated to enable comparison with the authentic integrity metric supplied by the trusted third party. In this first embodiment, the integrity metric is acquired by the measurement function **41** by generating a digest of the BIOS instructions in the BIOS memory and the SPK code. The measured integrity metric is signed using the trusted device **24** private key to provide confidence that the integrity metric has been acquired by the trusted device **24**. Such an acquired integrity metric, if verified as described above, gives a potential user of the platform **10** a high level of confidence that the platform **10** has not been subverted at a hardware, or BIOS program, level.

[**0054**] The measurement function **41** has access to: non-volatile memory **3** for storing a hash program **354** and a private key **355** of the trusted device **24**, and volatile memory **4** for storing acquired integrity metric in the form of a digest **361**. In appropriate embodiments, the volatile memory **4** may also be used to store the public keys and associated ID labels **360***a*-**360***n* of one or more authentic smart cards (not shown) that can be used to gain access to the platform **10**.

[**0055**] In one preferred implementation, as well as the digest, the integrity metric includes a Boolean value, which is stored in volatile memory **4** by the measurement function **31**, for reasons described below.

[**0056**] A process for acquiring an integrity metric for the computer platform **10** as used in first embodiments of the invention will now be described with reference to **FIG. 5**.

[**0057**] In step **500**, at switch-on, the measurement function **41** monitors the activity of the main processor **21** on the data, control and address lines (**26**, **27** & **28**) to determine whether the trusted device **24** is the first memory accessed. Processor **21** is directed to the trusted device **24**, which acts as a memory. In step **505**, if the trusted device **24** is the first memory accessed, in step **510**, the measurement function **41** writes to volatile memory **3** a Boolean value which indicates that the trusted device **24** was the first memory accessed. Otherwise, in step **515**, the measurement function writes a Boolean value which indicates that the trusted device **24** was not the first memory accessed.

[**0058**] In the event the trusted device **24** is not the first accessed, there is of course a chance that the trusted device **24** will not be accessed at all. This would be the case, for example, if the main processor **21** were manipulated to run the BIOS program first. Under these circumstances, the platform would operate, but would be unable to verify its integrity on demand, since the integrity metric would not be available. Further, if the trusted device **24** were accessed after the BIOS program had been accessed, the Boolean value would clearly indicate lack of integrity of the platform.

[**0059**] However, if a user is prepared to trust the BIOS the computer platform **10** can be arranged to use the BIOS instructions as the first instructions accessed.

[**0060**] In step **520**, when (or if) accessed as a memory by the main processor **21**, the main processor **21** reads the stored native hash instructions **354** from the measurement function **41** in step **525**. The hash instructions **354** are passed for processing by the main processor **21** over the data bus **26**. In step **530**, main processor **21** executes the hash instructions **354** and uses them, in step **535**, to compute a digest of the BIOS memory **29**, by reading the contents of the BIOS memory **29** and processing those contents according to the hash program. In step **540**, the main processor **21** writes the computed digest **361** to the appropriate non-volatile memory location **4** in the trusted device **24**. In a similar manner the measurement function **41** initiates the calculation of a digest for the SPK **32** that is correspondingly stored in an appropriate non-volatile memory location **4** in the trusted device **24**. The measurement function **41**, in step **545**, then calls the BIOS firmware **28** in the BIOS memory **29**, and execution continues, as described below.

[**0061**] Clearly, there are a number of different ways in which the integrity metric of the platform may be calculated, depending upon the scope of the trust required. The measurement of the BIOS program's integrity provides a fun-

damental check on the integrity of a platform's underlying processing environment. The integrity metric should be of such a form that it will enable reasoning about the validity of the boot process—the value of the integrity metric can be used to verify whether the platform booted using the correct BIOS. Optionally, individual functional blocks within the BIOS could have their own digest values, with an ensemble BIOS digest being a digest of these individual digests. This enables a policy to state which parts of BIOS operation are critical for an intended purpose, and which are irrelevant (in which case the individual digests must be stored in such a manner that validity of operation under the policy can be established).

[0062] Other integrity checks could involve establishing that various other devices, components or apparatus attached to the platform are present and in correct working order. In one example, the BIOS programs associated with a SCSI controller could be verified to ensure communications with peripheral equipment could be trusted. In another example, the integrity of other devices, for example memory devices or co-processors, on the platform could be verified by enacting fixed challenge/response interactions to ensure consistent results. Where the trusted device **24** is a separable component, some such form of interaction is desirable to provide an appropriate logical binding between the trusted device **24** and the platform. Also, although in the present embodiment the trusted device **24** utilises the data bus as its main means of communication with other parts of the platform, it would be feasible, although not so convenient, to provide alternative communications paths, such as hardwired paths or optical paths. Further, although in the present embodiment the trusted device **24** instructs the main processor **21** to calculate the integrity metric in other embodiments, the trusted device itself is arranged to measure one or more integrity metrics.

[0063] Preferably, the BIOS boot process includes mechanisms to verify the integrity of the boot process itself. Such mechanisms are already known from, for example, Intel's draft "Wired for Management baseline specification v 2.0—BOOT Integrity Service", and involve calculating digests of software or firmware before loading that software or firmware. Such a computed digest is compared with a value stored in a certificate provided by a trusted entity, whose public key is known to the BIOS. The software/firmware is then loaded only if the computed value matches the expected value from the certificate, and the certificate has been proven valid by use of the trusted entity's public key. Otherwise, an appropriate exception handling routine is invoked.

[0064] Optionally, after receiving the computed BIOS digest, the trusted device **24** may inspect the proper value of the BIOS digest in the certificate and not pass control to the BIOS if the computed digest does not match the proper value. Additionally, or alternatively, the trusted device **24** may inspect the Boolean value and not pass control back to the BIOS if the trusted device **24** was not the first memory accessed. In either of these cases, an appropriate exception handling routine may be invoked.

[0065] Further details of the operation of first embodiments of the invention will now be described with reference to FIGS. **6** to **8**.

[0066] Optionally, as shown in **FIG. 6**, to provide control and support to the computer platform **10** a system management counsel (SMC) **60** is coupled to computer platform **10** via connection **62**. In one embodiment, SMC **60** includes separate independent processors (not shown), such as standard non-networked personal computers (PCs). Connection **62** can include serial interfaces (e.g., RS-232 and USB), and/or private LAN connections. SMC **60** is primarily employed to authenticate SPK **32** during computer platform **10** initialization. In addition, computer platform **10** is configured via SMC **60**. In one embodiment, SMC **60** performs remote debugging for SPK **32** and SPGS **33**.

[0067] In one embodiment, GUI interfaces for system control and management are only implemented on SMCs **60**. This embodiment permits development and testing of system management interfaces and human factors in parallel with development of the rest of computer platform **10**, without having to wait for the entire computer platform **10** to be brought up.

[0068] More than one SMC **60** can be coupled to computer platform **10** via serial interface and/or LAN connection **62**. In one embodiment, SMC **60** functions are integrated into SPGS **33** in a computer platform **10** having a single processor, such as a workstation.

[0069] Additionally, the trust device **24** could be located in the SMC and act as the trusted device remotely to the computer platform **10**.

[0070] Once the trusted device **24** has initiated a trusted boot-up sequence, as described above, it is still necessary to ensure the chain of trust is maintained through to the initialisation of the operating domains. Therefore, in addition to utilising the trusted device **24** to provide information as to whether the computer platform can be trusted it is necessary to determine that a users operating environment can be trusted.

[0071] Accordingly, once the trusted device **24** has passed control to the BIOS firmware **28** the SPA **31** is arranged to provide a trusted operating environment as described below.

[0072] Initially on passing control to the BIOS firmware **28** the BIOS firmware **28**, inter alia, boots up and authenticates the EFI.

[0073] An EFI file system stores a secure platform (SP) loader, a system configuration database (SCD), a SPK image **32**, and a SPGS image **33**.

[0074] The EFI loads SP loader from EFI file system into memory **25**. The EFI authenticates this image using the processor **21** manufacturer's public key. This authentication requires that SP loader be digitally signed with the processor **21** manufacturer's private key.

[0075] The EFI then transfers control to SP loader stored in memory **25**. SP loader is an EFI-based secondary loader which is secure platform specific. SP loader is responsible for loading SP images into memory **25**.

[0076] In one embodiment, it is possible for execution to be transferred to an EFI shell prompt to enable initial system installation and other administrative details, which breaks the SP chain of trust. In this case, the EFI recognizes that trust was lost and does not precede with loading SP loader. Instead, computer platform **10** resets so that all processors **21** will again start fetching instructions from trusted device **24**.

[0077] SP loader running from memory **25** loads the SCD from EFI file system into memory **25**. SP loader then authenticates SCD employing a public key contained in the SP loader image. SP loader employs SCD to determine which SPK **32** and SPGS **33** images to load from EFI file system into memory. SP loader employs the above public key for authenticating the SPK **32** and SPGS **33** images. SP loader creates a virtual mapping for an entry area of SPK **32** with read and execute only permissions. SP loader then switches to virtual mode and branches to the SPK **32** entry point.

[0078] In the boot sequence for bringing up SPK **32**, SPK **32** running from memory **25** on processor **21**, initialises privilege state (e.g., interruption vector table (NT), control registers, and some interrupt configuration) and creates any other additional memory mappings required for SPK **32**, such as writeable areas for SPK data. SPK **32** then creates any required memory mappings and any additional set up required to run SPGS **33**.

[0079] A secure platform (SP) **34** mirrored file system stores two redundant control block images. SPK **32** reads the two redundant control block images from SP mirrored file system into SPK **32** in memory **25** as redundant control block images. The two redundant control block images contain control information initialized at the very first computer platform **10**. The redundant control block images are employed to test whether computer platform **10** has already been initialized.

[0080] In one embodiment, the redundant control block images each contain at least three distinct control areas. First control area contains an image that also is signed by the processor **21** manufacturer's public key, which was written when computer platform **10** booted for the first time. First control area is employed to store a root system key (RSK) in second control area. Second control area contains the RSK encrypted under itself. Second control area is employed to validate that a correct RSK has been supplied on subsequent boots. Encrypting the RSK under itself permits validation of the RSK, by comparing the results with the value already stored in second control area. Third control area contains a top-level directory of platform control information, including keys, pseudo random number generator (PRNG) state, and last entropy pool snapshot, all encrypted and integrity checked by the RSK.

[0081] SPK **32** typically has minimal or no I/O capability. In one embodiment the SP loader performs I/O accesses prior to transfer of control to SPK **32**. In another embodiment, SPGS **33** is brought up to an I/O ready state prior to the I/O operation to read from the disk, and returns control to SPK **32**. In another embodiment, SPGS **33** loads memory **25** and then a call is made to SPK **32** which performs the above operation.

[0082] SPK **32** determines whether the control areas of the two redundant control block images agree and the digital signature checks. If the control areas disagree, the control areas of the redundant control block image whose integrity checks as valid are used, and the control areas of the other redundant control block whose integrity checks as invalid are restored to match the used control areas of the valid redundant control block image. If the control areas of both redundant control block images are damaged, logs are used to recover, similar to many database systems, and to restore the control areas of both redundant control block images. Once the RSK is obtained, the boot process continues.

[0083] SPK **32** reads and decrypts protection keys from the SP mirrored file system.

[0084] The initial SPGS **33** domain initializes and performs discovery of I/O to include access to SMC **60**. The initial SPGS **33** domain loads an encrypted SCD from the SP mirrored file system. The initial SPGS **33** domain requests SPK **32** to decrypt the encrypted SCD. The decrypted SCD specifies the number of SPGS **33** domains to create and which system resources belong to which SPGS **33** domain. The initial SPGS **33** domain then creates each additional SPGS **33** domain specifying the corresponding subset of system resources to include in the processor **21** in which the SPGS **33** domain is run on.

[0085] Each SPGS **33** domain similarly reads the decrypted SCD and creates the specified domains. Each SPGS created domain includes the following. System resources are allocated to each SPGS **33** domain on a per domain basis. A domain initial image (DII) is loaded from EFI file system into memory **25** as DII. DII is typically an operating system specific loader for initiating the loading of an operating system for a specific domain in PL**2**. If SCD indicates that the given SPGS **33** domain is a secure domain, the self-contained public key of SP loader is employed to authenticate DII. Thus, DIIs which are to run in secure SPGS **33** domains are preferably digitally signed with the SP loader's private key. One use of a non-secure SPGS **33** domain is to allow development and debugging of DIIs.

[0086] On creation of each of the specified domains an associated virtual trusted device is created in the SPK **32**.

[0087] As the virtual trusted devices **37** are executed in the SPK **32**, which runs at the PL**0** level the only level that executes privileged instructions, the virtual trusted devices **37** can effectively be isolated from software executed in the other processor privilege levels. Accordingly, as the SPK **32** is trusted code a user can be confident that the virtual trusted devices are shielded from non-trusted software.

[0088] Each virtual trusted device **37** comprises, as shown in **FIG. 7**, a central routine **70** for controlling the overall operation of the virtual trusted device; a measurement function **71** for acquiring an integrity metric for an associated operating environment and obtaining the integrity metric acquired by the trusted device **24** and makes measurements on software that is to be executed in the associated operating environment; a cryptographic function **72** for signing, encrypting or decrypting specified data. Additionally, each virtual trusted device **37** is able to verify the integrity metric acquired by the trusted device **24** using the trusted third parties public key. The virtual trusted devices **37** have access to memory associated with the PL**0** level. Additionally, each virtual trusted device **37** is arranged to be isolated from any other virtual trusted device **37** that is associated with a separate operating environment.

[0089] On creation of an associated operating environment in PL**1** the associated virtual trusted device **37** in PL**0** is issued with a certificate that is associated with the user of the operating environment.

[0090] Each virtual trusted devices **37** certificate is stored in local memory in the PL**0** level. The certificate contains a

public key of the respective virtual trusted device **37** and, optionally, an authenticated value of an integrity metric for measured by a trusted third party to allow verification of the integrity metric acquired by the trusted device **24**. The certificate is signed by the trusted third party, using the trusted third parties private key, prior to the certificate being stored in the virtual trusted device **37**, thereby confirming that the trusted third party vouches for the virtual trusted device **37**. In this embodiment possible trusted third parties could be either the physical trusted device **24** or the SMC **60**.

[**0091**] As described below, a user, on accessing a virtual trusted device **37** associated with the respective operating environment, can obtain the computer platform integrity metric acquired and signed by the trusted device **24** with the trusted device's **24** private key and the integrity metric measured and signed by the virtual trusted device **37** and the virtual trusted device's **37** private key for the respective operating environment. Accordingly, the user is able to obtain all the integrity metric information required to allow verification that the respective operating environment can be trusted from the virtual trusted device **37** without the user needing to access the trusted device **24** directly.

[**0092**] As virtual trusted devices **37** are created and destroyed on the creation and destruction of operating environments it is necessary to ensure that their transitory existence does not compromise the trustworthiness of either the computer platform **10** or associated operating environments. As such, to ensure that trust can be maintained it is essential that secrets associated with the virtual trusted device(s) **37** do not exist in more than one active trusted device at any given time. This requires that strict and reliable methods in the computer platform **10** ensure that on the creation and destruction of a virtual trusted device **37** only one copy of relevant secrets (e.g. for example private keys) are maintained.

[**0093**] As such, destruction of a virtual trusted device **37** requires the permanent, safe, secret destruction of the virtual trusted devices secrets. If a virtual trusted device **37** is to be stored for re-use at a later date it secrets must be safely and secretly preserved for future use.

[**0094**] The secrets belonging to the virtual trusted device **37** could be stored in the physical trusted device **24** or SMC **60** using the protected storage facilities of a trusted platform module, for example. Virtual trusted device **37** secrets can be safely stored using the trusted computer platform association (TPCA) maintenance process.

[**0095**] For operating environments that need to continue to exist despite the computer platform **10** having to be power down and back up again it is possible to reassemble the stored associated virtual trusted device **37**. This allows the same virtual trusted device **37** to be maintained for the same operating environment, despite the temporary closing down of the operating environment.

[**0096**] However, the method required to reassemble a virtual trusted device **37** depends on the method used to dismantle the initial virtual trusted device **37**.

[**0097**] If a virtual trusted device **37** has been saved using the TCPA maintenance process, as described in section 7.3 of the TCPA specification, a new virtual trusted device **37** and trusted platform (i.e. operating environment) must be created (e.g. new endorsement key, credentials can be provided via the virtual trusted devices certificate). The TCPA maintenance process is used to transfer the appropriate secrets of the virtual trusted device to the new virtual trusted device **37** in the new operating environment. This is a two-step process, requiring first that the owner/user of the new operating environment check that the new virtual trusted device **37** and operating environment have at least the same level of security as the original virtual trusted device **37** and operating environment, such that the existing credentials do not overstate the security properties of the new virtual trusted device **37** and associated operating environment.

[**0098**] If the previous virtual trusted device **37** has been saved in full, a blank virtual trusted device **37** and associated operating environment are created in PL**0** and PL**1** respectively and the original secrets stored from the original virtual trusted device **37** are loaded into the new virtual trusted device. As above, the new operating environment must be checked that the new virtual trusted device **37** and operating environment have at least the same level of security as the original virtual trusted device **37** and associated operating environment, such that the existing credentials do not overstate the security properties of the new virtual trusted device **37** and operating environment. If a SMC **60** holds the secrets, some separate security service is required to confidentially communicate the secrets from the SMC **60** to the computer platform **10**. This will require a key distribution service, as is well known to a person skilled in the art.

[**0099**] This allow multiple operating environments to be created, where each operating environment has its own associated virtual trusted device **37** such that each virtual trusted device **37** derives the integrity metric for the computer platform **10** from the trusted device **24** and additionally measures an integrity metric for the associated operating environment. This allows a computer platform **10** to have multiple users, each with their own respective operating environment, where each operating environment is isolated from each other and each operating environment can provide an integrity metric for both itself and the computer platform **10**. This allows a user of an operating environment to determine whether his respective operating environment can be trusted without requiring any information as to whether any other operating environment is running on the computer platform **10**.

[**0100**] Additionally, as each domain is isolated and the virtual trusted devices **37** are executed in a privileged processor level PL**0** rouge software executed in one domain can not attack software executed in another domain.

[**0101**] **FIG. 8** illustrates a computer platform **10** having a trusted device **24** with BIOS and SPK code installed on processor **21**. The computer platform **10** is acting as a server having three operating environments **80'**, **80"**, **80'"** executed in privilege level **1** where each user would typically communicate with the operating environment **80'**, **80"**, **80'"** via a network connection. Each of the operating environments **80'**, **80"**, **80'"** has their own respective virtual trusted device **37'**, **37"**, **37'"** executed in the SPK **32** at privilege level PL**0**. Each virtual trusted device **37'**. **37"**, **37'"** has their own unique certificate (not shown) for their respective operating environment. When a user for each operating environment **80'**, **80"**, **80'"** wishes to communicate with their respective operating environment **80'**, **80"**, **80'"** they create a nonce (not

shown), such as a random number, and, issue a challenge to their respective virtual trusted device **37'**, **37"**, **37'"**. The nonce is used to protect the user from deception caused by replay of old but genuine signatures (called a 'replay attack') by untrustworthy platforms. The process of providing a nonce and verifying the response is an example of the well-known 'challenge/response' process.

[0102] The respective virtual trusted device **37'**, **37"**, **37'"** receives the challenge and creates an appropriate response. This may be a digest of the measured integrity metric of the computer platform integrity metric received from the trusted device **24** and signed with the trusted device's **24** private key and the measured integrity metric for the respective operating environment **80'**, **80"**, **80'"** signed with the respective virtual trusted device's **37** private key and the nonce, and optionally its ID label. The respective trusted device **37'**, **37"**, **37'"** return the signed integrity metric, accompanied by the respective virtual trusted devices **37'**, **37"**, **37'"** certificate and the trusted device's **24** certificate **350**, to the user.

[0103] The user receives the challenge response and verifies the certificate using the well known public key of the TP(s). The user then extracts the virtual trusted device's **37'**, **37"**, **37'"** public key and the trusted device's **24** public key from the certificate and uses them to decrypt the signed integrity metrics from the challenge response. Then the user verifies the nonce inside the challenge response. Next the user compares the computed integrity metrics, which it extracts from the challenge response, with the proper platform integrity metrics, which in this embodiment are extracted from the certificates. If any of the foregoing verification steps fails the whole process ends with no further communications taking place.

[0104] Assuming all is well the user and the trusted platform use other protocols to set up secure communications for other data, where the data from the platform is preferably signed by the trusted device **37'**, **37"**, **37'"** without any knowledge of the other two operating environments installed on the computer platform **10**.

[0105] A second set of embodiments will now be described with reference to FIGS. **9** to **11**. These second embodiments have many features in common with the first embodiments: differences are indicated positively; some common features are indicated positively, and the skilled person will appreciate how other features of the first embodiments may be applied in the second embodiments.

[0106] The second set of embodiments use a different form of virtualization technology. Alternative virtualization technologies will now be described. As the skilled person will appreciate, although only specific examples are provided, the principles of the present invention may be applied across the full range of virtualization technologies.

[0107] The basic requirement for virtualization is that any machine instruction that is either privileged or sensitive (including input and output) can be intercepted by a control layer (the virtualization layer). Instructions might be ones that would allow direct access to the real hardware or reveal sensitive state about other software running on it.

[0108] This virtualization layer can be achieved in different ways. Some processors (as has been described for the first embodiments) are naturally virtualizable, meaning that all privileged or sensitive instructions on that processor (generally a CPU) can be intercepted. Some CPUs (for example those according to the Intel IA-32 architecture) are not naturally virtualizable.

[0109] Most forms of CPU and 10 (input/output) device virtualization make use of the hardware protection facilities (such as privilege levels) provided by the real CPU that is being virtualized. On naturally virtualizable platforms that usually just means relying on the CPU protection mechanisms to make sure that the underlying virtualization layer always remains in control of the real hardware. This is all as has been described for the first embodiments above.

[0110] On CPUs which are not naturally a more software based approach needs to be taken. It may also be desirable to take such an approach on naturally virtualizable CPUs for performance reasons. This approach involves rewriting the operating systems running on top of the control layer so that they do not contain any privileged or sensitive instructions that would not naturally cause a transition to the control layer. "Para-virtualization" is the term used to describe such source code rewrites of the operating systems. The Xen virtual machine monitor relies on this approach. VMWare uses dynamic binary modification of the running operating systems as its means of remaining in control of the real hardware platform.

[0111] In a composite approach, both Intel (Vanderpool) and AMD (Pacifica) have developed hardware features in CPUs to support the running of virtual machine monitors. With this support any instruction, regardless of privilege level, can be made to trigger a transition to the control layer. With this approach it is no longer necessary to rely on running operating systems at different privilege levels to those on which they would normally run.

[0112] A general depiction of a platform using virtualization is provided in **FIG. 9**. Hardware **91** supports a virtual machine monitor (VMM) **92**—this is the virtualization layer which controls virtualization. On top of this are separate virtual machines with their own operating systems **93** and applications **94**.

[0113] Second embodiments of the invention will be described with reference to this depiction. The trusted device **24** is at the hardware level, and the isolation of a trusted module—and any other virtual component—is achieved by the virtualization layer **92**. The result is shown in **FIG. 10**: the hardware layer **91** contains a trusted device **95**, and trust routines **96** sit over the virtualization layer along with the associated operating environments. The second embodiments described below do not rely on the virtualization technology used.

[0114] For these second embodiments, the computer system of **FIG. 1**, the motherboard of **FIG. 2** and the trusted device of **FIG. 4** are all used essentially as described above, modified to be independent of the virtualization technology—further discussion on elements and use of a trusted device in accordance with TCG practice are however discussed below. Processor **21** in **FIG. 2** may therefore be any processor suitable for use with a virtualization technology, rather than specifically one with multiple privilege levels as specifically described above and as described in **FIG. 3**. Integrity metrics are obtained in the general manner indicated in **FIG. 5**—the measurement process for such metrics is not reliant on the form virtualization technology as

described (although relevant aspects of the virtualization technology may be measured by integrity metrics, as is described below). The measurement of integrity metrics in accordance with TCG practice is described more fully in, for example, "Trusted Computing Platforms—TCPA Technology in Context", edited by Siani Pearson, 2003, Prentice Hall PTR—integrity metrics as described in Pearson may be used in second embodiments of the invention.

[0115] The description of **FIG. 4** above describes the trusted device—and similarly trusted routines—each as a single entity. The approach taken in TCG specifications is to consider—rather than a single trusted device—multiple entities. The measurement engine which makes the first measurement within the trusted platform is termed the Root of Trust for Measurement (RTM)—this, in addition to the TPM, serves as a root of trust, generally achieved by a trusted source of such software vouching for it. A Core Root of Trust for Measurement (CRTM) comprises executable instructions that, when controlling the main processing engine on the platform, cause the platform to perform the functions of an RTM. The Trusted Platform Module (TPM) is an engine that that stores and reports measurements loaded into it by the RTM and subsequent measurement agents. The Trusted Building Block (TBB) comprises the CRTM and the connections between the CRTM, the TPM and the platform (including the platform's main processing engine). The Trusted Platform Module is, for the physical platform, comprised in a physical device. Second embodiments of the present invention describe how trust routines providing these elements in a virtualized form with respect to the relationships between a trust routine and the operating environment to which it relates can be related to the trusted elements of the physical platform itself.

[0116] **FIG. 5** indicates a process for obtaining an integrity metric of a trusted platform. In the second embodiments, integrity metrics are obtained in essentially this way, but may more specifically be produced the same way as a TCG RTM and TPM acquire an integrity metric. Similarly, supply of an integrity metric may be achieved in the same way that a TCG TPM supplies an integrity metric. One such metric is a measurement of a trust routine—such an integrity metric includes the RTM, CRTM, TPM and TBB for the trust routine, each providing a virtualized equivalent to these elements of TCG trusted platforms. Measurement and provision of integrity metrics according to TCG teaching, together with further description of these entities in TCG implementations, may be found in Pearson as referenced above. This applies equally to trust routines—in second embodiments, a trust routine acquires and supplies an integrity metric of the operating environment in the same way that a TCG RTM and TPM acquire and supply an integrity metric.

[0117] It should be appreciated that the trusted device and trust routines can have normal TCG Attestation Identities. An Attestation Identity, as described in Pearson, is a statistically unique, difficult to forge or counterfeit, identity which is verifiable to either a local or a remote entity. It comprises a digital certificate as a public part which contains a label and a public key all signed by a trusted entity and a private key retained as a secret in the TPM. Such an identity provides sufficient evidence that a trusted platform contains the capabilities and data that must be trustworthy if reports about the software environment in that platform are to be trusted. For the physically embodied TPM, such identities (there may be any number) are generated by the TPM and attested by a trusted entity termed a Privacy-CA. The Privacy-CA attests to the Attestation Identity by creating a certificate (termed an AIK Certificate) as described—this binds the identity key to the identity label and generic information about the platform. The Privacy-CA is persuaded to provide such attestation by being provided with sufficient information to establish that the trusted platform containing the TPM is a genuine trusted platform. In TCG specifications, this is provided by a Trusted Platform Module Entity vouching that a TPM is genuine by installing an endorsement key pair in the TPM and embedding the public key of the pair in an endorsement credential; and by a platform manufacturer using a Platform Entity to provide an analogous Platform Certificate to prove that the platform is a genuine trusted platform. In embodiments, there may also be required a Conformance Credential provided by the manufacturer to show that the design of TPM and the design of the platform meet TCG specifications. For the purpose of discussion of these second embodiments, it will be assumed that the trusted device has a conventional TCG Attestation Identity—discussion which follows relates to mechanisms which enable trust routines to obtain Attestation Identities of this type.

[0118] The following features are provided by second embodiments of the invention:

[0119] The server per se is measured via the physical platform's RTM and store the result in its physical TPM.

[0120] The server makes measurements of each virtual trusted platform, including at least:

[0121] The virtual TPM (a digest of the executable instructions that customise an engine that instantiates the virtual TPM). This includes the public Endorsement Key [pubEK] of that virtual TPM.

[0122] The virtual TBB (a digest of the executable instructions that customise an engine that instantiates the virtual TBB, which includes the virtual RTM).

[0123] The virtual normal platform.

[0124] A virtual trusted platform is measured via its virtual RTM and stores the results in its virtual TPM.

[0125] In order for a virtual TPM to obtain an Attestation Identity, the server's physical TPM must have previously used an Attestation Identity of its own to

[0126] sign a virtual Endorsement Credential, which comprises:

[0127] measurements from the server, including measurements of the virtualisation layer.

[0128] measurements of the virtual TPM in the virtual trusted platform

[0129] sign a virtual Platform Credential, which comprises:

[0130] measurements from the server, including measurements of the virtualisation layer.

[0131] measurements of the virtual TBB in the virtual trusted platform

[0132] When a virtual trusted platform reports its measurements, the virtual trusted platform's virtual TPM uses the virtual platform's Attestation Identity to sign the measurements.

[0133] When a virtual trusted platform is stopped, the virtual TPM credentials are secured using the physical platform's TPM and binding to the integrity measurements of the virtual trusted platform's TBB

[0134] The steps involved in enabling a virtual trusted platform to acquire an Attestation Identity are shown in **FIG. 11**.

[0135] As indicated above, the trusted device may use (1110) conventional TCG procedures and protocols to obtain an Attestation Identity. The AIK certificate could be a conventional TCG AIK certificate. In alternative embodiments, however, the AIK Certificate may be a modified TCG AIK certificate, whose "security properties" field comprises a description of the platform, including its virtualisation processes that create the trust routines and operating environments. These extra fields could be part of the "security properties" field in the platform's Platform Credential, supplied to the entity creating the AIK Certificate.

[0136] In order for a virtual TPM to obtain an Attestation Identity, the server's physical TPM must have previously used an Attestation Identity of the physical of the server to sign a virtual Endorsement Credential (1120). This is consistent with, but goes beyond, the first embodiments, which require that the trusted device uses one of its private asymmetric keys to vouch for a public key belonging to a trust routine.

[0137] In such second embodiments, when a platform instantiates a trust routine, the trust routine comprises the virtual equivalent of a TCG TPM containing a TCG Endorsement Key. Since the platform fully controls the trust routine, the platform is fully capable of obtaining that Endorsement Key from the trust routine. The platform is a credible entity to construct and sign a certificate attesting that that EK belongs to a properly constructed (virtual) TPM created by the platform, this certificate functioning as an Endorsement Credential.

[0138] The certificate is thus a conventional TCG Endorsement Credential, signed by a TCG Attestation Identity belonging to the trusted device. This is possible provided the AIK credential belonging to the trusted device has fields comprising a description of the platform's security properties, including its virtualisation processes and at least part of its trust routines.

[0139] In alternative embodiments, the certificate is a modified TCG Endorsement Credential, with extra fields comprising: (1) metrics obtained by the trusted device describing the platform, including its virtualisation processes; (2) metrics obtained by the trusted device describing at least part of the trust routine. The certificate is as before signed by a TCG Attestation Identity belonging to the trusted device.

[0140] In order for a virtual TPM to obtain an Attestation Identity, the server's physical TPM must have previously used an Attestation Identity of the physical of the server to sign a virtual Platform Credential (1130). When a platform instantiates a trust routine, the trust routine comprises the virtual equivalent of a TCG TBB and TPM. The trust routine and its associated operating environment contain a TCG TPM, TBB, RTM, and CRTM securely bound to a virtual normal platform, this certificate functioning as a Platform Credential.

[0141] The platform is a credible entity to construct and sign a certificate attesting that there exists a properly constructed (virtual) TCG trusted platform containing a particular TPM.

[0142] The certificate is a conventional TCG Platform Credential, signed by a TCG Attestation Identity belonging to the trusted device. This possible provided the AIK credential belonging to the trusted device has fields comprising a description of the platform's security properties, including its virtualisation processes and at least part of its trust routines.

[0143] In alternative embodiments, this certificate is a modified TCG Platform Credential, whose extra fields comprise (1) metrics obtained by the trusted device describing the platform, including its virtualisation processes that create the trust routines and operating environments; (2) metrics obtained by the trusted device describing at least part of the trust routine. The certificate is as before signed by a TCG Attestation Identity belonging to the trusted device.

[0144] The virtual TPM of the trust routine is now equipped with the necessary credentials to obtain an Attestation Identity Certificate (1140). This is consistent with, but goes beyond, the first embodiments which indicate that a trusted party may use one of its private asymmetric keys to vouch for a public key belonging to a trust routine.

[0145] When a trust routine creates an Attestation Identity, it may wish to acquire a TCG Attestation Identity Certificate for that identity. The trust routine follows conventional TCG procedures and protocols, and supplies the Endorsement Credential and Platform Credential created by the platform.

[0146] It should be appreciated that a CA (distinguished by TCG as a "Privacy CA") can follow normal TCG procedures and protocols to create an Attestation Identity certificate for a trust routine. The certificate, however, should be a modified AIK certificate whose extra fields comprise an indication of the TCG Attestation Identity belonging to the trusted device.

[0147] Alternative embodiments are possible as the platform is itself a credible entity to construct and sign an Attestation Identity certificate for the trust routine without following TCG procedures and protocols. This is because the platform fully controls the trust routine. Such a certificate is a modified TCG Attestation identity certificate, whose extra fields comprise: (1) metrics obtained by the trusted device describing the platform, including its virtualisation processes that create the trust routines and operating environments; (2) metrics obtained by the trusted device describing at least part of the trust routine. The certificate is signed by a TCG Attestation Identity belonging to the trusted device.

[0148] When in possession of an Attestation Identity certificate, the trust routine can act as a conventional TCG TPM. In particular, when a trust routine supplies integrity

metrics about an operating environment, the trust routine simply follows conventional TCG procedures and protocols, and signs the metrics using an Attestation Identity key.

[0149] In second embodiments of the invention, interactions between entities—from third parties to entities associated with the platform to users—may thus be essentially consistent with TCG procedures and protocols.

What is claimed:

1. A computer apparatus for creating a trusted environment comprising a trusted device arranged to acquire a first integrity metric to allow determination as to whether the computer apparatus is operating in a trusted manner; a processor arranged to allow execution of a first trust routine and associated first operating environment, and means for restricting access of the first operating environment to resources available to the trust routine, wherein the trust routine is arranged to acquire a second integrity metric to allow determination as to whether the first operating environment is operating in a trusted manner.

2. A computer apparatus as claimed in claim 1, wherein the means for restricting access of the first operating environment comprises a control layer of software and an operating system of the first operating environment adapted such that any instructions in the operating system of the first operating environment with potential to affect any environment outside the first operating environment cause a transition to the control layer.

3. A computer apparatus according to claim 2, wherein the trusted device is a tamper resistant device.

4. A computer apparatus according to claim 2, wherein the trust routine is arranged to incorporate cryptographical functionality for restricting access to data associated with the trust routine.

5. A computer apparatus according to claim 1, wherein the trusted device has an attestation identity and an attestation identity certificate containing attestation by a third party that the trusted device comprises a valid trusted platform module and that the computer apparatus comprises a valid trusted platform.

6. A computer apparatus according to claim 5, wherein the trust routine has an endorsement credential providing attestation by the trusted device that the trust routine comprises a valid virtual trusted platform module and a platform credential providing attestation by the trusted device that the trust routine and the first operating environment comprises a valid virtual trusted platform.

7. A computer apparatus according to claim 6, wherein the trust routine has an attestation identity and an attestation identity certificate containing attestation that the trusted device comprises a valid trusted platform module and that the computer apparatus comprises a valid trusted platform.

8. A computer apparatus according to claim 7, wherein the attestation is provided by a third party.

9. A computer apparatus according to claim 7, wherein the attestation is provided by the trusted device.

10. A computer apparatus according to claim 7, wherein the trusted device is arranged on powering down of the computer apparatus to store the endorsement credential, the platform credential and the attestation identity certificate of the trust routine.

11. A method for creating a trusted environment comprising acquiring a first integrity metric to allow determination as to whether a computer apparatus is operating in a trusted manner; executing a first trust routine and an associated first operating environment, restricting the first operating environment's access to resources available to the trust routine, and arranging the trust routine to acquire a second integrity metric to allow determination as to whether the first operating environment is operating in a trusted manner.

12. A method according to claim 11, wherein the first integrity metric is acquired by a trusted device in the computing apparatus, the trusted device having an attestation identity attested by a third party.

13. A method according to claim 12, further comprising the trusted device providing the trust routine with an endorsement credential providing attestation by the trusted device that the trust routine comprises a valid virtual trusted platform module and a platform credential providing attestation by the trusted device that the trust routine and the first operating environment comprises a valid virtual trusted platform.

14. A method according to claim 13, further comprising the trust routine generating an attestation identity and obtaining an attestation identity certificate using the endorsement credential and the platform credential.

15. A data structure comprising an attestation identity certificate for a trusted device in computer apparatus, the attestation identity certificate comprising at least a public key, a label, and a description of the computer apparatus including its virtualization processes to enable the trusted device to provide credentials for trust routines, all being signed by a trusted party.

16. A data structure comprising an attestation identity certificate for a trust routine running on computer apparatus having a trusted device, the attestation identity certificate comprising at least a public key and a label, all being signed by a trusted party.

17. A data structure according to claim 16, wherein the trusted party is a third party and the attestation identity certificate further comprises an indication of the attestation identity of the trusted device.

18. A data structure according to claim 16, wherein the trusted party is the trusted device and the attestation identity certificate further comprises a description of the computer apparatus including its virtualization processes and a description of the trust routine.

* * * * *